Positive Focusing is Directly Useful

Jui-Hsuan Wu (Ray) and Beniamino Accattoli

LIX, Ecole Polytechnique & Inria Saclay

Proofs and Algorithms Seminar
LIX, Ecole Polytechnique & Inria Saclay, Palaiseau, France

June 13th 2024

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

$$t, u ::= x \mid tu \mid \lambda x.t$$

In a call-by-value setting, general applications tu become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

$$t, u ::= x \mid tu \mid \lambda x.t$$

In a call-by-value setting, general applications tu become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

$$t, u ::= x \mid tu \mid \lambda x.t \mid \texttt{let}\ x\ \texttt{=}\ u\ \texttt{in}\ t$$

In a call-by-value setting, general applications *tu* become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x \leftarrow u] \text{ (explicit substitution)}$$

In a call-by-value setting, general applications *tu* become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x{\leftarrow}u] \text{ (explicit substitution)}$$

In a call-by-value setting, general applications *tu* become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

tu

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (x\llbracket u \rrbracket)[x \leftarrow \llbracket t \rrbracket]$$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{[\![\cdot]\!]} (x[\![u]\!])[x\leftarrow[\![t]\!]]$$

value as the left subterm
of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{array}{lll} & \xrightarrow{[\![\cdot]\!]} & (x[\![u]\!])[x\leftarrow[\![t]\!]] \\ tu & \xrightarrow{[\![\cdot]\!]} & ([\![t]\!]x)[x\leftarrow[\![u]\!]] \end{array}$$

value as the left subterm
of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (x\llbracket u \rrbracket)[x \leftarrow \llbracket t \rrbracket]$$

value as the left subterm of an application

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (\llbracket t \rrbracket x)[x \leftarrow \llbracket u \rrbracket]$$

value as the right subterm of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{[\![\cdot]\!]} (x[\![u]\!])[x{\leftarrow}[\![t]\!]]$$

value as the left subterm of an application

$$tu \xrightarrow{[\![\cdot]\!]} ([\![t]\!]x)[x{\leftarrow}[\![u]\!]]$$

value as the right subterm of an application

$$tu \xrightarrow{[\![\cdot]\!]} (xy)[x{\leftarrow}[\![t]\!]][y{\leftarrow}[\![u]\!]]$$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{[\![\cdot]\!]} (x[\![u]\!])[x\leftarrow[\![t]\!]]$$

value as the left subterm of an application

$$tu \xrightarrow{[\![\cdot]\!]} ([\![t]\!]x)[x\leftarrow[\![u]\!]]$$

value as the right subterm of an application

$$tu \xrightarrow{[\![\cdot]\!]} (xy)[x\leftarrow[\![t]\!]][y\leftarrow[\![u]\!]]$$

values as both subterms of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd}
 & (x[\![u]\!])[x\leftarrow[\![t]\!]] \\
tu \arrow[ur, bend left, "[\![\cdot]\!]"] \arrow[r, "[\![\cdot]\!]"] \arrow[dr, bend right, "[\![\cdot]\!]"'] & ([\![t]\!]x)[x\leftarrow[\![u]\!]] \\
 & (xy)[x\leftarrow[\![t]\!]][y\leftarrow[\![u]\!]]
\end{tikzcd}$$

- $(x[\![u]\!])[x\leftarrow[\![t]\!]]$: value as the left subterm of an application
- $([\![t]\!]x)[x\leftarrow[\![u]\!]]$: value as the right subterm of an application
- $(xy)[x\leftarrow[\![t]\!]][y\leftarrow[\![u]\!]]$: values as both subterms of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{[\![\cdot]\!]} (x[\![u]\!])[x\leftarrow[\![t]\!]]$$

value as the left subterm of an application

$$tu \xrightarrow{[\![\cdot]\!]} ([\![t]\!]x)[x\leftarrow[\![u]\!]]$$

value as the right subterm of an application

$$tu \xrightarrow{[\![\cdot]\!]} (xy)[x\leftarrow[\![t]\!]][y\leftarrow[\![u]\!]]$$

values as both subterms of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

$$xy \longrightarrow$$

substituting zw for x

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd}[column sep=huge] & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] & \text{value as the left subterm of an application} \\ tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] & \text{value as the right subterm of an application} \\ & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket] & \text{values as both subterms of an application} \end{tikzcd}$$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

$$xy \xrightarrow[\text{substituting } zw \text{ for } x]{\qquad\qquad\qquad} (zw)y$$

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (x\llbracket u \rrbracket)[x \leftarrow \llbracket t \rrbracket]$$

value as the left subterm of an application

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (\llbracket t \rrbracket x)[x \leftarrow \llbracket u \rrbracket]$$

value as the right subterm of an application

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (xy)[x \leftarrow \llbracket t \rrbracket][y \leftarrow \llbracket u \rrbracket]$$

values as both subterms of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

$$xy \xrightarrow[\text{substituting } zw \text{ for } x]{\times} (zw)y$$

Classifications of call-by-value calculi with ESs

It is possible to go further by restricting the immediate sub-terms of applications to be variables instead of values.

This gives us nine different forms of applications:
the general form tu and eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$
vs.
$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Classifications of call-by-value calculi with ESs

It is possible to go further by restricting the immediate sub-terms of applications to be variables instead of values.

This gives us nine different forms of applications:
the general form tu and eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{I}\mathsf{I}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}\mathsf{I})[x{\leftarrow}\mathsf{I}] \to \mathsf{I}\mathsf{I}$$

- Variables as values?

Classifications of call-by-value calculi with ESs

It is possible to go further by restricting the immediate sub-terms of applications to be variables instead of values.

This gives us nine different forms of applications:
the general form tu and eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Classifications of call-by-value calculi with ESs

It is possible to go further by restricting the immediate sub-terms of applications to be variables instead of values.

This gives us nine different forms of applications:
the general form tu and eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Classifications of call-by-value calculi with ESs

It is possible to go further by restricting the immediate sub-terms of applications to be variables instead of values.

This gives us nine different forms of applications:
the general form tu and eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Classifications of call-by-value calculi with ESs

It is possible to go further by restricting the immediate sub-terms of applications to be variables instead of values.

This gives us nine different forms of applications:
the general form tu and eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Positive Focusing is Directly Useful

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one often has the following subsitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow\lambda z.t] \to (y(\lambda z.t))[x\leftarrow\lambda z.t]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future.
$\hookrightarrow$ non-useful

Some more examples:

- $(xy)[x\leftarrow\lambda z.t] \to ((\lambda z.t)y)[x\leftarrow\lambda z.t]$ is useful
- $x[x\leftarrow\lambda z.t] \to (\lambda z.t)[x\leftarrow\lambda z.t]$ is non-useful
- $(xx)[x\leftarrow\lambda z.t]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one often has the following subsitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow\lambda z.t] \to (y(\lambda z.t))[x\leftarrow\lambda z.t]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future.
$\hookrightarrow$ non-useful

Some more examples:

- $(xy)[x\leftarrow\lambda z.t] \to ((\lambda z.t)y)[x\leftarrow\lambda z.t]$ is useful
- $x[x\leftarrow\lambda z.t] \to (\lambda z.t)[x\leftarrow\lambda z.t]$ is non-useful
- $(xx)[x\leftarrow\lambda z.t]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one often has the following subsitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow\lambda z.t] \to (y(\lambda z.t))[x\leftarrow\lambda z.t]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future.
$\hookrightarrow$ non-useful

Some more examples:

- $(xy)[x\leftarrow\lambda z.t] \to ((\lambda z.t)y)[x\leftarrow\lambda z.t]$ is useful
- $x[x\leftarrow\lambda z.t] \to (\lambda z.t)[x\leftarrow\lambda z.t]$ is non-useful
- $(xx)[x\leftarrow\lambda z.t]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one often has the following subsitution rule:

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x{\leftarrow}\lambda z.t] \to (y(\lambda z.t))[x{\leftarrow}\lambda z.t]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future.
$\hookrightarrow$ non-useful

Some more examples:

- $(xy)[x{\leftarrow}\lambda z.t] \to ((\lambda z.t)y)[x{\leftarrow}\lambda z.t]$ is useful
- $x[x{\leftarrow}\lambda z.t] \to (\lambda z.t)[x{\leftarrow}\lambda z.t]$ is non-useful
- $(xx)[x{\leftarrow}\lambda z.t]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one often has the following subsitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow\lambda z.t] \to (y(\lambda z.t))[x\leftarrow\lambda z.t]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future.
$\hookrightarrow$ non-useful

Some more examples:

- $(xy)[x\leftarrow\lambda z.t] \to ((\lambda z.t)y)[x\leftarrow\lambda z.t]$ is useful
- $x[x\leftarrow\lambda z.t] \to (\lambda z.t)[x\leftarrow\lambda z.t]$ is non-useful
- $(xx)[x\leftarrow\lambda z.t]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one often has the following subsitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow\lambda z.t] \to (y(\lambda z.t))[x\leftarrow\lambda z.t]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future.
$\hookrightarrow$ non-useful

Some more examples:

- $(xy)[x\leftarrow\lambda z.t] \to ((\lambda z.t)y)[x\leftarrow\lambda z.t]$ is (directly) useful
- $x[x\leftarrow\lambda z.t] \to (\lambda z.t)[x\leftarrow\lambda z.t]$ is non-useful
- $(xx)[x\leftarrow\lambda z.t]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one often has the following subsitution rule:

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x{\leftarrow}\lambda z.t] \to (y(\lambda z.t))[x{\leftarrow}\lambda z.t]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future.
$\hookrightarrow$ non-useful

Some more examples:

- $(xy)[x{\leftarrow}\lambda z.t] \to ((\lambda z.t)y)[x{\leftarrow}\lambda z.t]$ is (directly) useful
- $x[x{\leftarrow}\lambda z.t] \to (\lambda z.t)[x{\leftarrow}\lambda z.t]$ is non-useful
- $(xx)[x{\leftarrow}\lambda z.t]$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\lambda z.t] \to (\lambda z.t)[x{\leftarrow}\lambda z.t]$ is non-useful
  while $x[x{\leftarrow}\lambda z.t]y \to (\lambda z.t)[x{\leftarrow}\lambda z.t]y$ is useful
- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!
- Renaming chains:

$$
\begin{array}{ll}
 & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\
\to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\
\to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}]
\end{array}
$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\lambda z.t] \to (\lambda z.t)[x{\leftarrow}\lambda z.t]$ is non-useful
  while $x[x{\leftarrow}\lambda z.t]y \to (\lambda z.t)[x{\leftarrow}\lambda z.t]y$ is useful
- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!
- Renaming chains:

$$\begin{array}{rl} & (x_0 t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0 t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0 t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x\leftarrow\lambda z.t] \to (\lambda z.t)[x\leftarrow\lambda z.t]$ is non-useful
  while $x[x\leftarrow\lambda z.t]y \to (\lambda z.t)[x\leftarrow\lambda z.t]y$ is useful

- Indirect usefulness:
  $(xy)[x\leftarrow z][z\leftarrow \mathsf{I}] \to (xy)[x\leftarrow \mathsf{I}][z\leftarrow \mathsf{I}]$
  $\hookrightarrow$ It is useful!

- Renaming chains:

$$\begin{array}{rl} & (x_0 t)[x_0\leftarrow x_1][x_1\leftarrow x_2]\cdots[x_{k-1}\leftarrow x_k][x_k\leftarrow \mathsf{I}] \\ \to & (x_0 t)[x_0\leftarrow x_1][x_1\leftarrow x_2]\cdots[x_{k-1}\leftarrow \mathsf{I}][x_k\leftarrow \mathsf{I}] \\ \to^* & (x_0 t)[x_0\leftarrow \mathsf{I}][x_1\leftarrow \mathsf{I}]\cdots[x_{k-1}\leftarrow \mathsf{I}][x_k\leftarrow \mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x \leftarrow \lambda z.t] \to (\lambda z.t)[x \leftarrow \lambda z.t]$ is non-useful
  while $x[x \leftarrow \lambda z.t]y \to (\lambda z.t)[x \leftarrow \lambda z.t]y$ is useful
- Indirect usefulness:
  $(xy)[x \leftarrow z][z \leftarrow \mathsf{I}] \to (xy)[x \leftarrow \mathsf{I}][z \leftarrow \mathsf{I}]$
  $\hookrightarrow$ It is useful!
- Renaming chains:

$$\begin{array}{ll} & (x_0 t)[x_0 \leftarrow x_1][x_1 \leftarrow x_2] \cdots [x_{k-1} \leftarrow x_k][x_k \leftarrow \mathsf{I}] \\ \to & (x_0 t)[x_0 \leftarrow x_1][x_1 \leftarrow x_2] \cdots [x_{k-1} \leftarrow \mathsf{I}][x_k \leftarrow \mathsf{I}] \\ \to^* & (x_0 t)[x_0 \leftarrow \mathsf{I}][x_1 \leftarrow \mathsf{I}] \cdots [x_{k-1} \leftarrow \mathsf{I}][x_k \leftarrow \mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\lambda z.t] \to (\lambda z.t)[x{\leftarrow}\lambda z.t]$ is non-useful
  while $x[x{\leftarrow}\lambda z.t]y \to \underline{(\lambda z.t)}[x{\leftarrow}\lambda z.t]\underline{y}$ is useful
- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!
- Renaming chains:

$$
\begin{array}{rl}
 & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\
\to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\
\to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}]
\end{array}
$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\lambda z.t] \to (\lambda z.t)[x{\leftarrow}\lambda z.t]$ is non-useful
  while $x[x{\leftarrow}\lambda z.t]y \to \underline{(\lambda z.t)}[x{\leftarrow}\lambda z.t]\underline{y}$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$ is useful or not?
  $\hookrightarrow$ It is useful!

- Renaming chains:

$$\begin{array}{ll} & (x_0 t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0 t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0 t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x\leftarrow\lambda z.t] \to (\lambda z.t)[x\leftarrow\lambda z.t]$ is non-useful
  while $x[x\leftarrow\lambda z.t]y \to \underline{(\lambda z.t)}[x\leftarrow\lambda z.t]\underline{y}$ is useful
- Indirect usefulness:
  $(xy)[x\leftarrow z][z\leftarrow \mathsf{I}] \to (xy)[x\leftarrow \mathsf{I}][z\leftarrow \mathsf{I}] \to (\mathsf{I}y)[x\leftarrow \mathsf{I}][z\leftarrow \mathsf{I}]$
  $\hookrightarrow$ It is useful!
- Renaming chains:

$$
\begin{array}{rl}
 & (x_0\,t)[x_0\leftarrow x_1][x_1\leftarrow x_2]\cdots[x_{k-1}\leftarrow x_k][x_k\leftarrow \mathsf{I}] \\
\to & (x_0\,t)[x_0\leftarrow x_1][x_1\leftarrow x_2]\cdots[x_{k-1}\leftarrow \mathsf{I}][x_k\leftarrow \mathsf{I}] \\
\to^* & (x_0\,t)[x_0\leftarrow \mathsf{I}][x_1\leftarrow \mathsf{I}]\cdots[x_{k-1}\leftarrow \mathsf{I}][x_k\leftarrow \mathsf{I}]
\end{array}
$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\lambda z.t] \to (\lambda z.t)[x{\leftarrow}\lambda z.t]$ is non-useful
  while $x[x{\leftarrow}\lambda z.t]y \to \underline{(\lambda z.t)}[x{\leftarrow}\lambda z.t]\underline{y}$ is useful
- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}] \to (\mathsf{I}y)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is (indirectly) useful!
- Renaming chains:

$$\begin{array}{ll} & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\lambda z.t] \to (\lambda z.t)[x{\leftarrow}\lambda z.t]$ is non-useful
  while $x[x{\leftarrow}\lambda z.t]y \to \underline{(\lambda z.t)}[x{\leftarrow}\lambda z.t]\underline{y}$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}] \to (\mathsf{I}y)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is (indirectly) useful!

- Renaming chains:

$$\begin{array}{ll} & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Positive Focusing is Directly Useful

Focusing

Focusing is a technique first introduced by Andreoli to reduce non-determinism in *logic programming* (or *proof search*) in linear logic.

It comes from a simple observation:

Rule	invertible	non-invertible
Phase	negative	positive
Connective	negative	positive

Focusing gives more structure to proofs.
↪ focused proofs can be seen as a (light) canonical form of proofs.

Focusing

Focusing is a technique first introduced by Andreoli to reduce non-determinism in *logic programming* (or *proof search*) in linear logic.

It comes from a simple observation:

Rule	invertible	non-invertible
Phase	negative	positive
Connective	negative	positive

Focusing gives more structure to proofs.
$\hookrightarrow$ focused proofs can be seen as a (light) canonical form of proofs.

Focusing

Focusing is a technique first introduced by Andreoli to reduce non-determinism in *logic programming* (or *proof search*) in linear logic.

It comes from a simple observation:

Rule	invertible	non-invertible
Phase	negative	positive
Connective	negative	positive

Focusing gives more structure to proofs.
$\hookrightarrow$ focused proofs can be seen as a (light) canonical form of proofs.

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^- and δ^+:

- δ^- yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^+ yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^- and δ^+:

- δ^- yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^+ yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^- and δ^+:

- δ^- yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^+ yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^{-} and δ^{+}:

- δ^{-} yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^{+} yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^{-} and δ^{+}:

- δ^{-} yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^{+} yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Positive Focusing is Directly Useful

Positive λ-calculus $\lambda_{\texttt{pos}}$

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened. Every term is of the form $E\langle x\rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathtt{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: $\lambda_{\texttt{pos}}$ (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened. Every term is of the form $E\langle x\rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \rightarrow_{\text{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened. Every term is of the form $E\langle x\rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{rl} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ & x[x{\leftarrow}w'_1 w'_1][w'_1{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened. Every term is of the form $E\langle x\rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x {\leftarrow} yz] \mid t[x {\leftarrow} \lambda y.u]$$

- ESs are flattened. Every term is of the form $E\langle x \rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}\underline{z}z']\underline{[z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]]} \\ \rightarrow_{\text{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \rightarrow_{\text{om}_+} & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x \leftarrow yz] \mid t[x \leftarrow \lambda y.u]$$

- ESs are flattened. Every term is of the form $E\langle x \rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \to_{\text{oe}_+} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \to_{\text{om}_+} & x[x \leftarrow w_1' w_1'][w_1' \leftarrow z' z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x\leftarrow yz] \mid t[x\leftarrow \lambda y.u]$$

- ESs are flattened. Every term is of the form $E\langle x\rangle$.
- Restricted form of explicit substitutions:
    1. Minimalistic application yz
    2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll}
 & x[x\leftarrow yy][y\leftarrow zz'][z\leftarrow \lambda w.w'[w'\leftarrow ww]] \\
\to_{\mathsf{oe}_+} & x[x\leftarrow yy][y\leftarrow (\lambda w.w'[w'\leftarrow ww])z'][z\leftarrow \lambda w.w'[w'\leftarrow ww]] \\
\to_{\mathsf{om}_+} & x[x\leftarrow w_1'w_1'][w_1'\leftarrow z'z'][z\leftarrow \lambda w.w'[w'\leftarrow ww]]
\end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened. Every term is of the form $E\langle x\rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{oem}_+} & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Explicit positive λ-calculus λ_{xpos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u] \mid t[x{\leftarrow}(\lambda y.u)z]$$

- ESs are flattened. Every term is of the form $E\langle x\rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{rl} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{om}_+} & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Explicit positive λ-calculus λ_{xpos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u] \mid \color{red}{t[x{\leftarrow}(\lambda y.u)z]}$$

- ESs are flattened. Every term is of the form $E\langle x\rangle$.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{om}_+} & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- multiplicative rule (m-rule) for firing a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- exponential rule (e-rule) for firing an ES (of values) and makes a substitution

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$
\begin{array}{rcl}
t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\
v & ::= & x \mid \lambda x.t
\end{array}
$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- multiplicative rule (m-rule) for firing a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- exponential rule (e-rule) for firing an ES (of values) and makes a substitution

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\texttt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\texttt{vsc}}$:

- multiplicative rule (m-rule) for firing a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- exponential rule (e-rule) for firing an ES (of values) and makes a substitution

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\texttt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x \leftarrow u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\texttt{vsc}}$:

- multiplicative rule (m-rule) for firing a β-redex and creates an ES

$$(\lambda x.t)u \to t[x \leftarrow u]$$

- exponential rule (e-rule) for firing an ES (of values) and makes a substitution

$$C\langle x \rangle[x \leftarrow v] \to C\langle v \rangle[x \leftarrow v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x \leftarrow u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- multiplicative rule (m-rule) for firing a β-redex and creates an ES

$$(\lambda x.t)u \to t[x \leftarrow u]$$

- exponential rule (e-rule) for firing an ES (of values) and makes a substitution

$$C\langle x\rangle[x \leftarrow v] \to C\langle v\rangle[x \leftarrow v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x \leftarrow u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- multiplicative rule (m-rule) for firing a β-redex and creates an ES

$$(\lambda x.t)u \to t[x \leftarrow u]$$

- exponential rule (e-rule) for firing an ES (of values) and makes a substitution

$$C\langle x\rangle[x \leftarrow v] \to C\langle v\rangle[x \leftarrow v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- multiplicative rule (m-rule) for firing a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- exponential rule (e-rule) for firing an ES (of values) and makes a substitution

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\texttt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x \leftarrow u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\texttt{vsc}}$:

- multiplicative rule (m-rule) for firing a β-redex and creates an ES

$$(\lambda x.t)u \to t[x \leftarrow u]$$

- exponential rule (e-rule) for firing an ES (of values) and makes a substitution

$$C\langle x\rangle[x \leftarrow v] \to C\langle v\rangle[x \leftarrow v]$$

Known result: the number of m-steps is a reasonable cost model.

Positive Focusing is Directly Useful

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate e-rules for variables ($\rightarrow_{\mathsf{e_{var}}}$) and abstractions ($\rightarrow_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful e-steps ($\rightarrow_{\mathsf{e_u}}$) from non useful e-steps ($\rightarrow_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e_{var}}} + \rightarrow_{\mathsf{e_u}}$

Non-useful reduction $= \rightarrow_{\mathsf{e_{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate e-rules for variables ($\rightarrow_{\mathsf{e_{var}}}$) and abstractions ($\rightarrow_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful e-steps ($\rightarrow_{\mathsf{e_u}}$) from non useful e-steps ($\rightarrow_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e_{var}}} + \rightarrow_{\mathsf{e_u}}$

Non-useful reduction $= \rightarrow_{\mathsf{e_{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate e-rules for variables ($\rightarrow_{\mathsf{e_{var}}}$) and abstractions ($\rightarrow_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful e-steps ($\rightarrow_{\mathsf{e_u}}$) from non useful e-steps ($\rightarrow_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e_{var}}} + \rightarrow_{\mathsf{e_u}}$

Non-useful reduction $= \rightarrow_{\mathsf{e_{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate *e*-rules for variables ($\to_{\mathsf{e_{var}}}$) and abstractions ($\to_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful *e*-steps ($\to_{\mathsf{e_u}}$) from non useful *e*-steps ($\to_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \to_{\mathsf{m}} + \to_{\mathsf{e_{var}}} + \to_{\mathsf{e_u}}$

Non-useful reduction $= \to_{\mathsf{e_{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate e-rules for variables ($\to_{\mathsf{e_{var}}}$) and abstractions ($\to_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful e-steps ($\to_{\mathsf{e_u}}$) from non useful e-steps ($\to_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \to_{\mathsf{m}} + \to_{\mathsf{e_{var}}} + \to_{\mathsf{e_u}}$

Non-useful reduction $= \to_{\mathsf{e_{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate *e*-rules for variables ($\rightarrow_{\mathsf{e_{var}}}$) and abstractions ($\rightarrow_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful *e*-steps ($\rightarrow_{\mathsf{e_{u}}}$) from non useful *e*-steps ($\rightarrow_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e_{var}}} + \rightarrow_{\mathsf{e_{u}}}$

Non-useful reduction $= \rightarrow_{\mathsf{e_{nu}}}$

Positive Focusing is Directly Useful

Big picture

$$\lambda_{\mathtt{ovsc}}$$

Big picture

$$\lambda_{\texttt{ovsc}}$$

$$t \longrightarrow^{*} u$$

Big picture

$$\lambda_{\mathtt{ovsc}}$$

t $\quad$ t' $\quad$ u

Big picture

$$\lambda_{\mathtt{ovsc}}$$

$$t \xrightarrow{\quad\text{Core } \lambda_{\mathtt{ovsc}}\quad}^{*} t' \xrightarrow{\text{Non-useful}}^{*} u$$

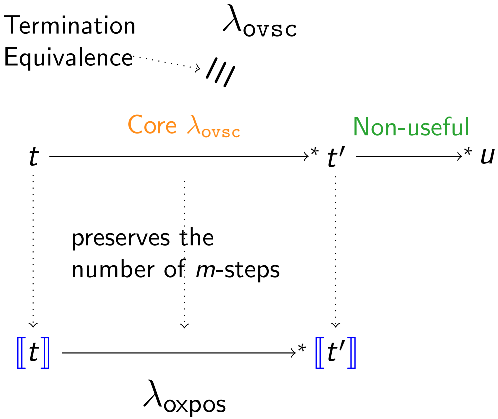
Big picture

Termination $\lambda_{\mathtt{ovsc}}$

Equivalence

$$t \xrightarrow{\text{Core } \lambda_{\mathtt{ovsc}}}{}^{*} t' \xrightarrow{\text{Non-useful}}{}^{*} u$$

Big picture


Termination
Equivalence
$\lambda_{\mathtt{ovsc}}$
Core $\lambda_{\mathtt{ovsc}}$
Non-useful
t
t'
u
preserves the
number of m-steps
$[\![t]\!]$
$[\![t']\!]$
λ_{oxpos}

Big picture

Termination
Equivalence

$\lambda_{\mathtt{ovsc}}$

$\equiv$

$$
\begin{tikzcd}
t \arrow[rr, "\text{Core } \lambda_{\mathtt{ovsc}}"] \arrow[d, dotted] \arrow[drr, phantom, "\equiv"] & & {}^{*}\, t' \arrow[r, "\text{Non-useful}"] \arrow[d, dotted] & {}^{*}\, u \\
\llbracket t \rrbracket \arrow[rr, "\lambda_{\mathsf{oxpos}}"'] & & {}^{*}\, \llbracket t' \rrbracket &
\end{tikzcd}
$$

Conclusion and Future work

- We show that the compactness of λ_{pos} allows one to capture the essence of usefulness. What is remarkable is that λ_{pos} is an outcome of a study of term representation inspired by focusing.
- Future work:
  1. efficient implementation of meta-level renamings involved in λ_{pos}. We expect this to be doable in an efficient way via an appropriate abstract machine.
  2. λ_{pos} for call-by-need evaluation

Conclusion and Future work

- We show that the compactness of $\lambda_{\mathtt{pos}}$ allows one to capture the essence of usefulness. What is remarkable is that $\lambda_{\mathtt{pos}}$ is an outcome of a study of term representation inspired by focusing.
- Future work:
  1. efficient implementation of meta-level renamings involved in $\lambda_{\mathtt{pos}}$. We expect this to be doable in an efficient way via an appropriate abstract machine.
  2. $\lambda_{\mathtt{pos}}$ for call-by-need evaluation

Conclusion and Future work

- We show that the compactness of λ_{pos} allows one to capture the essence of usefulness. What is remarkable is that λ_{pos} is an outcome of a study of term representation inspired by focusing.
- Future work:
  1. efficient implementation of meta-level renamings involved in λ_{pos}. We expect this to be doable in an efficient way via an appropriate abstract machine.
  2. λ_{pos} for call-by-need evaluation

Thank you for your attention!